使用Python实现"Go元编程"

翁伟

关于我

- 翁伟，原常驻新加坡，现居深圳
- 不断折腾（10年？）的全端程序员
  - .net -> python -> go
- 再次回国创业，觅伙伴
  - wengwei@xipintech.com

希品科技
xipintech.com

元编程

- 指某类计算机程序的编写，这类计算机程序编写或者操纵其它程序（或者自身）作为它们的数据
- 与手工编写全部代码相比，程序员可以获得更高的工作效率

元编程其实无处不在

- 最常见的元编程工具是编译器，它可以将程序员使用高级语言编写的相对短小的程序转换为等价的汇编语言或者机器语言程序
- 其它元编程系统则允许以编程方式操纵一种语言。宏系统即是这样一种简单的系统

C宏

```
#define list_for_each(pos, head) \
for (pos = (head)->next; pos != (head); pos = pos->next)
```

“Go，互联网时代的C，下一个C”

– 许式伟

希品科技
xipintech.com

Go元编程?

- 木有宏
- 木有泛型
- 反射很难用

- “gopher 这几年很努力，解决了许多别的语言中不存在的问题……” - @赖勇浩

看看很努力的pythoner + gopher
如何用py来实现go的泛型?

```go
package set

type StringSet struct {
    data map[string]struct{}
}

func (s *StringSet) Init() {
    s.data = make(map[string]struct{})
}

func (s *StringSet) Contains(v string) bool {
    _, ok := s.data[v]
    return ok
}

func (s *StringSet) Add(v string) {
    s.data[v] = struct{}{}
}

func (s *StringSet) Len() int {
    return len(s.data)
}

func (s *StringSet) Copy() *StringSet {
    res := NewStringSet()
    res.Union(s)
    return res
}
```

```go
package set

type IntSet struct {
    data map[int]struct{}
}

func (s *IntSet) Init() {
    s.data = make(map[int]struct{})
}

func (s *IntSet) Contains(v int) bool {
    _, ok := s.data[v]
    return ok
}

func (s *IntSet) Add(v int) {
    s.data[v] = struct{}{}
}

func (s *IntSet) Len() int {
    return len(s.data)
}

func (s *IntSet) Copy() *IntSet {
    res := NewIntSet()
    res.Union(s)
    return res
}
```

- StringSet与IntSet的代码很类似，仅是类型不同
- 泛型的典型使用场景

```go
type SomeTypeSet struct {
    data map[SomeType]struct{}
}

func (s *SomeTypeSet) Init() {
    s.data = make(map[SomeType]struct{})
}

func (s *SomeTypeSet) Contains(v SomeType) bool {
    _, ok := s.data[v]
    return ok
}

func (s *SomeTypeSet) Add(v SomeType) {
    s.data[v] = struct{}{}
}

func (s *SomeTypeSet) Len() int {
    return len(s.data)
}
```

```python
src = open("sometype_set.go").read()
for T in ["string", "int"]:
    f = open("%s_set.go" % T, "w+")
    f.write(src.replace("SomeTypeSet",
        "%sSet" % T.capitalize()).replace("SomeType", T))
    f.close()
```

简单、粗暴、有效

自动生成

- MakeFile?

Makefile

```
export GOPATH := $(shell pwd)
export PATH := ${PATH}:${GOPATH}/bin
export GOBIN := ${GOPATH}/bin

build:
    python gen_codes.py
    go install main
```

- 监控文件修改：https://github.com/gorakhargosh/watchdog

再来看看“ORM”

我们使用thrift来定义对象

```
struct UserProfile {
    1: string ID (
        label = "用户资料",
        baseURL = "/admin/portal/user"
        search = "Name"
        listedFields = "Name, Email"
    )
    2: required string Name (label="姓名")
    3: string MobilePhone (label="移动电话")
    4: string Email (label="电子邮箱")
    5: string HomeAddress (label="家庭地址")
    6: string PostCode (label="邮编")
    7: string Fax (label="传真")
}
```

用户资料

姓名*

移动电话

传真

电子邮箱

邮编

家庭地址

保存

取消

用户资料

新建

搜索...

搜索

	序号	姓名	电子邮箱	操作
	1	翁伟	wengwei@xipintech.com	编辑

当前显示所有 1 条记录　每页显示 20 条记录

同一对象元信息

- 增删改查
- 全文搜索
- 后台权限
- RPC

- 对象传递、赋值、显示需要大量相似的代码

```go
func (o *UserProfile) ReadForm(params map[string]string) (hasError bool) {
    if val, ok := params["Name"]; ok {
        o.Name = val
    }
    if val, ok := params["MobilePhone"]; ok {
        o.MobilePhone = val
    }
    if val, ok := params["Email"]; ok {
        o.Email = val
    }
    if val, ok := params["HomeAddress"]; ok {
        o.HomeAddress = val
    }
    if val, ok := params["PostCode"]; ok {
        o.PostCode = val
    }
    if val, ok := params["Fax"]; ok {
        o.Fax = val
    }
    return o.ValidateData()
}
```

```go
func (o *UserProfile) GetFieldAsString(fieldKey string) (Value string) {
    switch fieldKey {
    case "ID":
        Value = o.ID.Hex()
    case "Name":
        widget := o.NameWidget()
        if widget.Type == "select" {
            Value = widget.Val()
        } else if widget.GetBindData != nil {
            Value = widget.Val()
        } else {
            Value = o.Name
        }
    case "MobilePhone":
        widget := o.MobilePhoneWidget()
        if widget.Type == "select" {
            Value = widget.Val()
        } else if widget.GetBindData != nil {
            Value = widget.Val()
        } else {
            Value = o.MobilePhone
        }
    case "Email":
        widget := o.EmailWidget()
```

```go
func (o *UserProfile) Widgets() []*Widget {
    return []*Widget{
        o.NameWidget(),
        o.MobilePhoneWidget(),
        o.EmailWidget(),
        o.HomeAddressWidget(),
        o.PostCodeWidget(),
        o.FaxWidget(),
    }
}
```

```go
func (o *UserProfile) GetListedLabels() []*IDLabelPair {
    return []*IDLabelPair{
        &IDLabelPair{
            ID:    "Name",
            Label: "姓名",
        },
        &IDLabelPair{
            ID:    "Email",
            Label: "电子邮箱",
        },
    }
}
```

- 上述所有代码都必须根据对象元信息生成出来
- 生成的代码不会错

对象描述

- Thrift
  - 严谨
  - 可扩展
  - 跨语言

```
struct UserProfile {
    1: string ID (
        label = "用户资料",
        baseURL = "/admin/portal/user"
        search = "Name"
        listedFields = "Name, Email"
    )
    2: required string Name (label="姓名")
    3: string MobilePhone (label="移动电话")
    4: string Email (label="电子邮箱")
    5: string HomeAddress (label="家庭地址")
    6: string PostCode (label="邮编")
    7: string Fax (label="传真")
}
```

解析对象

- ptsd: https://github.com/wickman/ptsd
  - thrift lexer/parser using ply
- ply: http://www.dabeaz.com/ply/
  - An implementation of lex and yacc parsing tools for Python

代码生成模板

- http://www.cheetahtemplate.org

```
func (o *$obj.name.value) ReadForm(params map[string]string) (hasError bool) {
#for field in $obj.fields
    if val, ok := params["$field.name.value"]; ok {
    #if $field.type == "string"
        o.$field.name.value = val
    #else if $field.type == "i32"
        intVal, _ := strconv.Atoi(val)
        o.$field.name.value = int32(intVal)
    #else if $field.type == "double"
        floatVal, _ := strconv.ParseFloat(val, 64)
        o.$field.name.value = floatVal
    #else if $field.type == "bool"
        o.$field.name.value = (val != "" && val != "false")
    #else if $field.type == "list<string>"
        o.$field.name.value = strings.Split(val, "\n")
        for i, k := range o.$field.name.value {
            o.$(field.name.value)[i] = strings.Trim(k, "\r")
        }
    #end if
    }
#end if
#end for
    return o.ValidateData()
}
```

```go
func (o *UserProfile) ReadForm(params map[string]string) (hasError bool) {
    if val, ok := params["Name"]; ok {
        o.Name = val
    }
    if val, ok := params["MobilePhone"]; ok {
        o.MobilePhone = val
    }
    if val, ok := params["Email"]; ok {
        o.Email = val
    }
    if val, ok := params["HomeAddress"]; ok {
        o.HomeAddress = val
    }
    if val, ok := params["PostCode"]; ok {
        o.PostCode = val
    }
    if val, ok := params["Fax"]; ok {
        o.Fax = val
    }
    return o.ValidateData()
}
```

```
func (o *$obj.name.value) GetFieldAsString(fieldKey string) (Value string) {
    switch fieldKey {
#for field in $obj.fields
    case "$field.name.value":
    #if $field.name.value == "ID"
        Value = o.ID.Hex()
    #else if $field.type == "string"
        Value = o.$field.name.value
        #end if
    #else if $field.type == "double"
        Value = strconv.FormatFloat(o.$field.name.value, 'f', 2, 64)
    #else if $field.type == "bool"
        Value = strconv.FormatBool(o.$field.name.value)
    #end if
#end for
    }
    return
}
```

```go
func (o *UserProfile) GetFieldAsString(fieldKey string) (Value string) {
    switch fieldKey {
    case "ID":
        Value = o.ID.Hex()
    case "Name":
        widget := o.NameWidget()
        if widget.Type == "select" {
            Value = widget.Val()
        } else if widget.GetBindData != nil {
            Value = widget.Val()
        } else {
            Value = o.Name
        }
    case "MobilePhone":
        widget := o.MobilePhoneWidget()
        if widget.Type == "select" {
            Value = widget.Val()
        } else if widget.GetBindData != nil {
            Value = widget.Val()
        } else {
            Value = o.MobilePhone
        }
    case "Email":
        widget := o.EmailWidget()
```

```
func (o *$obj.name.value) Widgets() []*Widget {
    return []*Widget{
#for field in $obj.fields
#if $field.name.value != "ID"
        o.${field.name.value}Widget(),
#end if
#end for
    }
}
```

```go
func (o *UserProfile) Widgets() []*Widget {
    return []*Widget{
        o.NameWidget(),
        o.MobilePhoneWidget(),
        o.EmailWidget(),
        o.HomeAddressWidget(),
        o.PostCodeWidget(),
        o.FaxWidget(),
    }
}
```

```
func (o *$obj.name.value) GetListedLabels() []*IDLabelPair {
    return []*IDLabelPair{
        #for field in $obj.listedFields
        &IDLabelPair {
            ID : "$field.key",
            Label : "$field.label",
            #if hasattr(field, "order")
            Order : "$field.order",
            #end if
            #if hasattr(field, "widget")
            Widget: "$field.widget",
            #end if
        },
        #end for
    }
}
```

```go
func (o *UserProfile) GetListedLabels() []*IDLabelPair {
    return []*IDLabelPair{
        &IDLabelPair{
            ID:    "Name",
            Label: "姓名",
        },
        &IDLabelPair{
            ID:    "Email",
            Label: "电子邮箱",
        },
    }
}
```

```python
import sys
from ptsd.loader import Loader
from Cheetah.Template import Template

thrift_file = sys.argv[1]
out_path = sys.argv[2]

def main(thrift_file):
    loader = Loader(thrift_file)

    for module in loader.modules.values():
        for struct in module.structs:
            tpl = open('tmpl/go.tmpl', 'r').read().decode("utf8")
            t = Template(tpl, searchList=[{"obj": obj}])
            code = str(t)
            write_file(out_path + '/gen_' + obj.name.value.lower() + ".go", code)

main(thrift_file)
```

所以我们项目有~~30万~~80万行代码

```
     663 ./watch/scan.go
     184 ./watch/watchdir.go
     563 ./xuanwu/src/common.go
  836733 total
➜  zfw_code git:(develop) ✗ find . -name '*.go' | xargs wc -l
```

希品科技
xipintech.com

- 上述仅仅是“演示代码”
  - 实际代码要复杂得多：https://github.com/sipin/xuanwu
  - 但原理不变
- 仅供参考、切勿使用
  - 无文档
  - 不维护
  - 内部已推倒重来

如何定制?

- 生成一次，后面随便改
- 随模板修改反复生成:
  - 提供定制接口方法回调
  - 覆盖特定方法

```go
func UserProfileAddInsertCallback(cb func(obj IXuanWuObj)) {
    insertCB_UserProfile = append(insertCB_UserProfile, cb)
}

func UserProfileAddUpdateCallback(cb func(obj IXuanWuObj)) {
    updateCB_UserProfile = append(updateCB_UserProfile, cb)
}

func UserProfileAddExtraValidation(validate func(o *UserProfile) bool) {
    extraValidation_UserProfile = append(extraValidation_UserProfile, validate)
}
```

```go
package userprofile

import (
    "xuanwu"
    "app/models/userprofile"
)

func init() {
    //call back
    userprofile.UserProfileAddInsertCallback(func(obj xuanwu.IXuanWuObj) {
        sendNoticeToUser(obj.(*userprofile.UserProfile))
    })
}
```

“元编程”

- 适用于go，显然也适用于其它语言
  - Go编译速度快，不在乎代码量暴涨
- 业务中存在重复、相似的代码均可考虑“元编程”

- Python简单方便，资源多
  - 编译：ply
  - 模板：cheetah
  - 监控：watchdog

Q & A

其实

- Python不快
  - 比方说，体验过git的极速，就很难忍hg的“不快”
- 元编程最好还是解析语言本身
  - http://golang.org/pkg/go/parser/
- go generate
  - https://docs.google.com/document/d/1V03LUfjSADDooDMhe-_K59EgpTEm3V8uvQRuNMAEnjg